TC3041 Base de Datos Avanzadas
Laboratorio II Oracle

Objetivo: Reconocer y practicar los conceptos del modelo objeto relacional: manejo de reglas (triggers), uso de constructores adicionales en la definición de tipos y tablas (row, array y ref) y su uso en un producto manejador de base de datos.

Instrucciones:

- Ésta es una actividad en equipo.
- El equipo tiene dos opciones para desarrollar la actividad.
    - Opción 1. Puede usar cualquier manejador de base de datos que soporte funcionalidad avanzada completa o parcialmente de SQL99 (row, array, ref types, inheritance, triggers) tal como PostgresSQL u Oracle (Oracle10 y versiones posteriores) o cualquier otro producto que usted enqcuentre que soporta la funcionalidad mencionada.
    - Opción 2. Puede usar cualquier manejador relacional "tradicional" que no incluya la las características anteriores pero usted y su equipo deben de simular la funcionalidad (mapeos, consultas sobre jerarquías, reglas)
- En cualquiera de las dos opciones puede ser necesario agregar funcionalidad como stored-procediures, o codificación de algún servicio en lenguaje de programación complementaria a SQL dependiendo de la herramienta que Ud. eligió.
- Una interfaz simple para integrar toda la funcionalidad es suficiente. No invierta tiempo en interfaces sofisticadas.
- El esquema a implementar es el sistema de clubes-estudiuantes-profesores-departamentos indicado abajo
-
- The DB schema to implement is the Minimal MedicPatient System describe below

Entregables

- Código fuente documentado: Script con comentarios utilizado para crear la base de datos
- Una descripción de como construyo la aplicación por ejemplo componentes, archivos y organización de los mismos.
- Screen shots de las tablas con datos después de la creación de las tablas
- Para la carga de datos se sugiere seguir las indicaciones indicadas más adelante en este documento
- Revisión en presencia del profesor (algunas se revisaran en clase y otras en hora de asesoría)

Como nota si su opción es Oracle, Oracle provee funcionalidad como la siguiente:

- *Object Types* for creating user-defined types.
- *Object Tables* for creating objects from object types.
- *Type hierarchies* with inheritance and corresponding *table hierarchies*.
- *VARRAYS* and *Nested Tables* that allow a structured collection of data to be the type of a table column.
- *Refs* (or object references) that are used to store logical pointers to objects.

1. Escenario

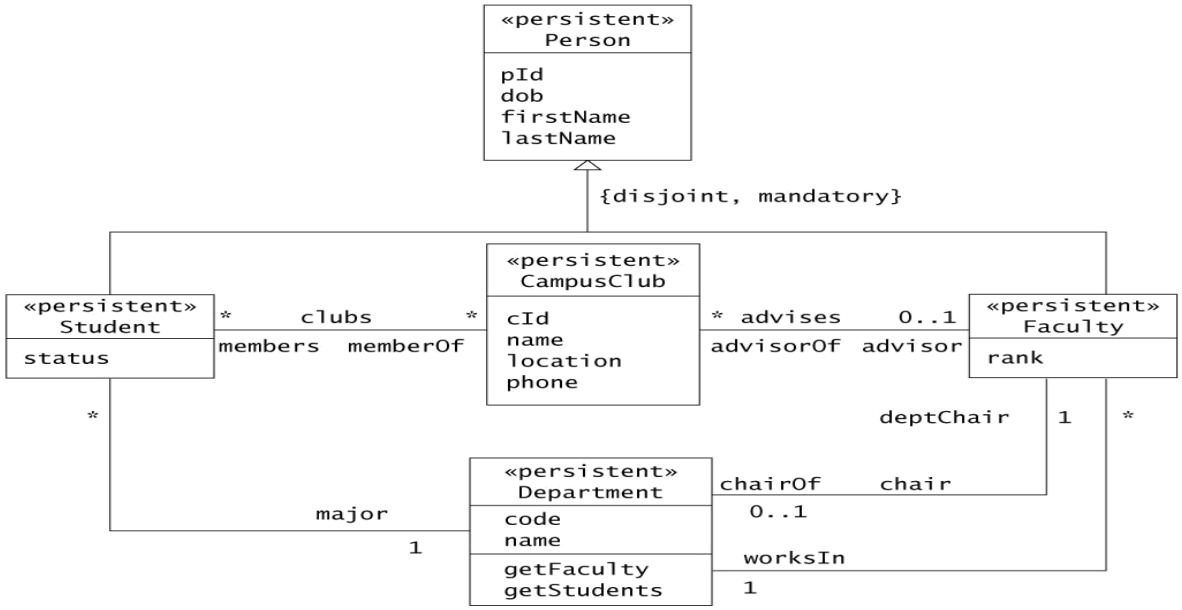
La base de datos que se implementará será la descrita por el siguiente diagrama. La Base de datos contiene información de estudiantes, profesores y departamentos académicos. Además de los clubes que existen en la universidad.

Los estudiantes y profesores deben de estar bajo la jerarquía de Persona. Los profesores pueden ser asesores de uno o varios clubes y trabajar para un departamento académico. Los estudiantes pertenecen a un departamento académico a través de su "major" y pueden pertenecer a uno o varios clubes. El club tiene como miembros a estudiantes y como asesor a un sólo profesor. Los departamentos académicos tienen como director a un profesor.

Se pide:

1. Creación de tipos y tablas
2. Poblar tablas (ver siguiete página)
3. Realizar consultas (ver siguiete página)

Diagrama de Clases UML para la aplicación The University Database Example



DDL en SQL99 para la creación de The University Database Example

Reglas de Negocio para The University Database Example

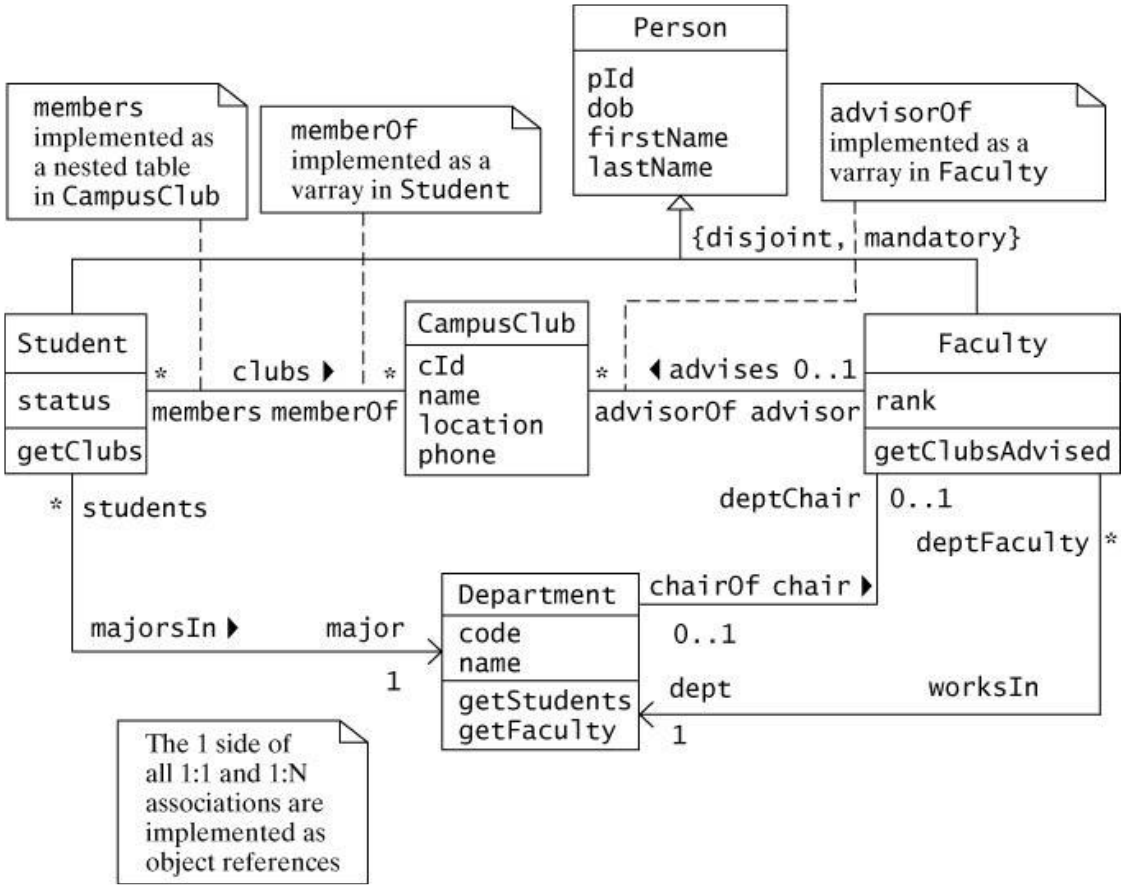
1. El rango (Rank) de un profesor es un indicador de su experiencia y trayectoria. Los valores posibles son: Instructor, Asistente, Asociado, y Titular
2. El estatus de un estudiante se indica por uno de los siguientes valores: freshman, sophomore, junior, senior
3. Un profesor (faculty member) puede ser director únicamente del departamento para el cual trabaja
4. Se debe de incrementar el salario de un profesor en un 10% cuando el profesor es promovido del rango de asistente a asociado
5. Si un profesor es director de departamento y se cambia de departamento entonces el departamento anterior debe quedar sin director.
6. Debe ser posible tener clubes con alumnos repetidos
7. Si un alumno se borra de la tabla de alumnos se debe eliminar de forma consistente su asociación a departamentos y clubes
8. No puede haber clubes con la misma dirección

Población de Datos
Insertar 30 alumnos
Insertar 6 profesores
Insertar dos departamentos
Tres profesores en departamento A
Tres profesores en departamento B
Insertar Tres clubes Club1, Club 2, Club 3
Insertar 10 alumnos en club1, 10 alumnos en club2, 15 alumnos en club 3
Al menos uno de los clubes debe tener alumnos que también participan en otro club
Debe de haber alumnos que NO participan en ningún club

Consultas mínimas

Alumnos en más de un club
Alumnos que no pertenecen a ningún club
Seleccionar primer nombre y fecha de nacimiento para alumnos que pertenecen a club 1
Encontrar al asesor académico del Club2 (nombre rango y departamento donde trabaja)

El siguiente diagrama detalla una estrategia para implementar la base de datos requerida.



```
create type personUdt as
(pid        varchar(11),
 firstName  varchar(20),
 lastName   varchar(20),
 dob            date)
 instantiable not final ref is system generated;

create type facultyUdt under personUdt as
( rank   varchar(10),
  advisorOf ref(campusClubUdt) scope campusClub array[5] references are checked on delete set
null,
  worksIn ref(departmentUdt) scope department references are checked on delete no action,
  chairOf ref(departmentUdt) scope department references are checked on delete set null)
 instantiable not final;
create type studentUdt under personUdt as
(  status varchar(10),
  memberOf ref(campusClubUdt) scope campusClub array[5] references are checked on delete set
null,
  major ref(departmentUdt) scope department references are checked on delete no action)
instantiable not final;

create table person of personUdt(primary key (pid), ref is personID system generated);
create table faculty of facultyUdt under person;
create table student of studentUdt under person;

create type departmentUdt as
 (  code varchar(3),
    name varchar(40),
    deptChair ref(facultyUdt) scope faculty referenced are checked on delete no action)
    instantiable not final ref is system generated
    method  getStudents() returns studentUdt array[1000],
    method getFaculty() returns facultyUdt array[50];

create table department of department_udt
( deptChair with options not null, primary key (code),  ref is departmentID system generated);

create type locationUdt as
(  street varchar(30),
    bldg varchar(5),
    room varchar(5)) not final;

create type campusClubUdt as
(  cId number,
   name      varchar(50),
   location locationUdt,
   phone  varchar(12),
   advisor ref(facultyUdt) scope faculty references are checked on delete cascade,
   members ref(studentUdt) scope student array[50] references are checked on delete set null)
   instantiable not final ref is system generated;

create table campusClub of campusClubUdt(primary key (cid), ref is campusClubId system
generated);
```